# Technical Breakdown of Position Based Fluid

Qingyuan Qie
will.qie@mail.utoronto.ca
University of Toronto
Toronto, ON, Canada

Changlin Su
sheldon.su@mail.utoronto.ca
University of Toronto
Toronto, ON, Canada

## ABSTRACT

This paper provides a technical breakdown of implementing a parallel version of the Position Based Fluids(PBF) method. We discussed the simulation algorithms of PBF, the effects of different confinements in the algorithms. We also go in depth of the data structures used to implement the algorithms efficiently. Due to the highly parallelizable nature of the algorithm, we provided a implementation using the CUDA platform which achieves realtime simulation with a particle count of 12k. A CPU version is also available.

## CCS CONCEPTS

• **Computing methodologies** → **Physical simulation**.

## KEYWORDS

Fluids Simulation, Physics-Based Animation

**ACM Reference Format:**
Qingyuan Qie and Changlin Su. 2021. Technical Breakdown of Position Based Fluid. In *Proceedings of ACM Conference (Conference'17).* ACM, New York, NY, USA, 3 pages. https://doi.org/0.0/0.0

## 1 INTRODUCTION

In Position Based Fluids(PBF) method, fluids are simulated with discrete particles, the surfaces of the fluids are reconstructed at render time based on the position of the underlying particles. In each time step, the PBF method first predicts the positions and velocity. Then it corrects the particle positions by enforcing the incompressibility constraints. The new velocity of the particles are computed from the new positions and the original positions of the particles.

Algorithm 1 is an overview of a simulation step in PBF method.

**Algorithm 1** simulation step

**for** all particles $i$ **do**: ▷ Fluid advect
  apply forces $\boldsymbol{v}_i = \boldsymbol{v}_i + \Delta t f_{ext}$
  predict position $\boldsymbol{x}_i^* = \boldsymbol{x}_i + \Delta t \boldsymbol{v_i}$
**end for**
**for** all particles $i$ **do**:
  find neighboring particles $F(\boldsymbol{x}_i^*)$
**end for**
**while** $iter < solverIterations$ **do**: ▷ Iterativly solves imcompressibility constraints
  **for** all particles $i$ **do**:
    calculate $\lambda_i$
  **end for**
  **for** all particles $i$ **do**:
    calculate $\Delta \boldsymbol{p}_i$
    perform collision detection and response
  **end for**
  **for** all particles $i$ **do**:
    update position $\boldsymbol{x}_i^* = \boldsymbol{x}_i^* + \Delta \boldsymbol{p}_i$
  **end for**
**end while**
**for** all particles $i$ **do**:
  update velocity $\boldsymbol{v}_i = \frac{1}{\Delta t}(\boldsymbol{x}_i^* - \boldsymbol{x}_i)$
  apply viscosity
  update position $\boldsymbol{x}_i = \boldsymbol{x}_i^*$
**end for**

## 2 ENFORCING INCOMPRESSIBILITY

For particle $i$ at position $p_i$, we compute the density of the fluid around particle $i$ using the estimator:

$$\rho_{F(i)} = \sum_{j \in F(i)} m_j W_{poly6}(\boldsymbol{p}_i - \boldsymbol{p}_j, h) \tag{1}$$

where $\rho_0$ is the rest density of the fluid, $m_j$ is the mass of the particle $j$, $h$ is a constant, and $W$ is the Poly6 kernel from [TODO: insert citation]. $F$ is the neighbor function that returns the neighboring particle of particle $i$.

The Poly6 kernel is defined as follows,

$$W_{poly6}(\boldsymbol{r}, h) = \frac{315}{64\pi h^9} \begin{cases} (h^2 - r^2)^3 & 0 \le r \le h \\ 0 & \text{otherwise} \end{cases}$$

where $r$ is the norm of $\boldsymbol{r}$. Combine equation (1) and the definition of poly6, we can notice that if the distance between particle $i$ and $j$ is greater than $h$, particle $j$ does not contribute to the density around particle $i$. Thus the neighbor-finding algorithm and need to find the particles whose distance to particle $j$ is smaller or equal to $h$, we will discuss the detail of the neighbor-finding algorithm later.

Then we introduce the constant density constraints $C_i$. For particle $i$, we have,

$$C_i(\boldsymbol{p}) = \frac{\rho_i}{\rho_0} - 1 \tag{2}$$

where $\rho_0$ is a constant that denotes the rest density of the fluid, and $\boldsymbol{p}$ is the position of all the particles in the system.

Since we want the density around each particle is always equal to $\rho 0$, we want to find a particle correction $\Delta(\boldsymbol{p})$ such that,

$$C(\boldsymbol{p} + \boldsymbol{\Delta(p)}) = 0 \tag{3}$$


*Conference'17, July 2017, Washington, DC, USA*

ACM ISBN 978-x-xxxx-xxxx-x/YY/MM…$15.00
https://doi.org/0.0/0.0

The solution to (3) is found by a series of Newton steps along the constraint gradient,

$$\Delta(\boldsymbol{p}) \approx \nabla C(\boldsymbol{p})\lambda \tag{4}$$

$$C(\boldsymbol{p} + \boldsymbol{\Delta(p)}) \approx C(\boldsymbol{p}) + \nabla C^T \Delta \boldsymbol{p} = 0 \tag{5}$$

$$\approx C(p) + \nabla C^T \nabla C \lambda = 0 \tag{6}$$

And the gradient of the constraint $C_i$ with respect to a particle $k$ is given by,

$$\nabla_{\boldsymbol{p}_k} C_i = \frac{1}{\rho_0} \sum_j \nabla_{\boldsymbol{k}} W(\boldsymbol{p}_i - \boldsymbol{p}_j, h) \tag{7}$$

Which has two different cases based on whether $k$ is a neighboring particle or $k$ is the particle $i$ itself,

$$\nabla_{\boldsymbol{p}_k} C_i = \frac{1}{\rho_0} \begin{cases} \sum_j \nabla_{\boldsymbol{p}_k} W(\boldsymbol{p}_i - \boldsymbol{p}_j, h) & \text{if } k = i \\ -\nabla_{\boldsymbol{p}_k} W(\boldsymbol{p}_i - \boldsymbol{p}_j, h) & \text{otherwise} \end{cases} \tag{8}$$

The original paper uses the Spiky kernel fot the gradient computation, which is defined as,

$$\nabla W_{spiky}(\boldsymbol{r}, h) = -\frac{45}{64\pi h^6} \begin{cases} (h-r)^2 \hat{\boldsymbol{r}} & 0 \le r \le h \\ 0 & \text{otherwise} \end{cases}$$

where $\hat{r}$ is the normalized $\boldsymbol{r}$.

Plug (8) into (6) and solves for $\lambda$ yields,

$$\lambda_i = -\frac{C_i(\boldsymbol{p})}{\sum_k |\nabla_{\boldsymbol{p}_k} C_i|^2} \tag{9}$$

Then the position correction $\Delta \boldsymbol{p}_i$ including affect from neighboring particles is

$$\Delta \boldsymbol{p}_i = \frac{1}{\rho_0} \sum_j (\lambda_i + \lambda_j) \nabla W(\boldsymbol{p}_i - \boldsymbol{p}_j, h) \tag{12}$$

And we have the position update formula,

$$\boldsymbol{p}_i^* = \boldsymbol{p}_i + \Delta \boldsymbol{p}_i \tag{13}$$

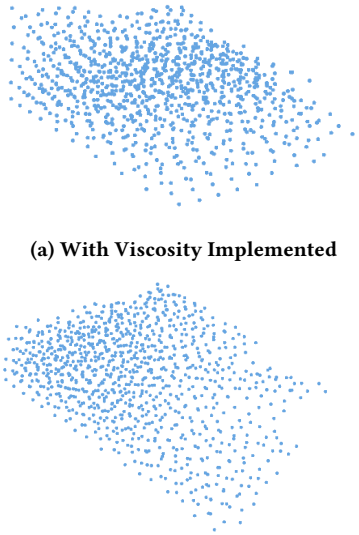
## 3 TENSILE INSTABILITY

The original PBF paper adds an artificial pressure to mitigate the issue of particle clamping. It introduced an additional correction coefficient defined as,

$$s_{corr} = -k \left( \frac{W(\boldsymbol{p}_i - \boldsymbol{p}_j, h)}{W(\Delta \boldsymbol{q}, h)} \right)^n \tag{14}$$

where $\Delta \boldsymbol{q}$ is a point some fixed distance inside the smoothing kernel radius and $k$ is a small positive constant. In our implementation, we take $k = 0.1$, $|\Delta \boldsymbol{q}| = 0.1h$ and $n = 4$. Then the modified position update formula becomes,

$$\Delta \boldsymbol{p}_i = \frac{1}{\rho_0} \sum_j (\lambda_i + \lambda_j + s_{corr}) \nabla W(\boldsymbol{p}_i - \boldsymbol{p}_j, h) \tag{15}$$



**(a) With Viscosity Implemented**

**(b) Without Viscosity Implemented**

**Figure 1: Comparison of the fluid motion with/without viscosity implemented, taken at the same time step.**

## 4 VORTICITY CONFINEMENT AND VISCOSITY

The original PBF paper implemented vorticity to replace lost energy. However, due to the time constraint, we did not implement vorticity confinement.

We implemented viscosity, which is defined as follows,

$$\boldsymbol{v}_i^* = \boldsymbol{v}_i + c \sum_j \boldsymbol{v}_{ij} \cdot W(\boldsymbol{p}_i - \boldsymbol{p}_j, h). \tag{16}$$

where $\boldsymbol{v}_{ij} = \boldsymbol{v}_j - \boldsymbol{v}_i$. The parameter $c$ is chosen to be 0.001 in our simulation after a few trail and error. The viscosity is important for convincing fluid motion. Figure 1 shows the effect of the viscosity. When viscosity is not implemented, the particles are more scattered and have the tendency of not moving together, which is not realistic. With viscosity implemented, the fluid is more "thick" visually and thus more similar to how real water moves.

## 5 BOUNDARY AND COLLISION RESPONSE

In our implementation, the fluid is confined in a cuboid whose surfaces are parallel to the coordinate planes. We deploy a simple collision scheme as describe in [TODO: CITATION]. Particles that goes out of bound in the fluid advect is clamped into the cuboid with an small distance to the boundary, and velocity of the particle is set to 0.

## 6 GRID-BASED NEIGHBOR FINDING

The Poly6 and Spiky function both return 0 when the distance of two particles is greater than $h$. Therefore, we adapted the grid-based neighbor finding algorithm described in [TODO: CITATION]. We implemented the CPU version ourselves, and we modified the source code of [TODO: CITATION] for CUDA version.

## 7 IMPLEMENTATION DETAILS

Table 1 shows the correspondence of algorithm steps to the name of the functions in our code. Since Algorithm 1 is highly parallelizable with the update of each particle can be computed in parallel, we launch one thread for each particle in the CUDA implementation.

**Table 1: Actual Function name to the pseudo-code in Algorithm 1**

| Line Number | CUDA Kernel Name CPU Function Name |
|---|---|
| 1 - 4, 14 | advect |
| 5 - 7 | build_grid |
| 9 - 11 | compute_lambda |
| 12-15 | compute_delta_position |
| 16-19 | update_position |
| 21 | update_velocity |
| 22 | viscosity_confinement |